Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления» (ИУ)
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии» (ИУ7)

Отчет по лабораторной работе №РК 1 по курсу «Анализ алгоритмов»

«Однопоточная и двупоточная реализация стандартного алгоритма поиска подстроки в строке»

Группа: ИУ7-53Б

Студент: _____ Дьяченко А. А.
(Подпись, дата) (Фамилия И. О.)

Преподаватель: _____ Строганов Д. В.
(Подпись, дата) (Фамилия И. О.)

Москва, 2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3
1 Аналитическая часть 4
1.1 Стандартный алгоритм 4
2 Конструкторская часть 5
2.1 Требования к ПО 5
2.2 Разработка алгоритмов 6
3 Технологическая часть 10
3.1 Средства реализации 10
3.2 Сведения о модулях программы 10
3.3 Реализация алгоритмов 11
4 Исследовательская часть 14
4.1 Технические характеристики 14
4.2 Пример работы программы 14
4.3 Время выполнения реализованных алгоритмов 16
ЗАКЛЮЧЕНИЕ 17
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 18

ВВЕДЕНИЕ

Задача поиска подстроки одна из достаточно распространённых в информатике. Строкой называют последовательность символов (в произвольном порядке) взятых из заданного алфавита. Так например, алфавитом могут быть цифры 0, 1 из которых можно составлять неограниченную по длине цепочку символов, например, 0110100110 или 0 [1].

Практическое значение задачи о точных совпадениях трудно преувеличить. Эта задача возникает в широком спектре приложений: текстовых редакторах, информационно-поисковых системах, электронных библиотеках, каталогах и справочниках и т. д. Алгоритмы поиска подстроки в строке также применяются при поиске заданных образцов в молекулах ДНК [2].

Целью данной лабораторной работы является создание ПО с последовательной однопоточной и двупоточной реализацией нахождения подстроки в строке стандартным алгоритмом.

Для достижения поставленной цели необходимо решить следующие задачи:

- описать стандартный алгоритм;
- определить средства программной реализации;
- реализовать последовательный и параллельный вариант стандартного алгоритма;
- сравнить и проанализировать реализации алгоритмов по затраченному времени;
- подготовить отчет по лабораторной работе.

1 Аналитическая часть

1.1 Стандартный алгоритм

Одним из самых очевидных и одновременно неэффективных алгоритмов является алгоритм последовательно (прямого) поиска. Суть его заключается в сравнении искомой подстроки с каждым словом в основной строке. Алгоритм не является эффективным, так как максимальное количество сравнений будет равно O((n-m+1)*m), где большинство из них на самом деле лишние. Для небольших строк поиск работает довольно быстро, но если в файлах с большим количеством информации последовательность символов будет искаться очень долго [4].

Однако, на скорость работы алгоритма можно повлиять, если использовать идею распараллеливания потоков. В данной работе реализован последовательный и параллельный подход. При параллельном используется два потока: один обрабатывает первую половину файла, второй — другую.

2 Конструкторская часть

В этом разделе будут представлены схемы и/или псевдокоды реализуемых алгоритмов.

2.1 Требования к ПО

К программе предъявляется ряд требований:

- искомая подстрока имеет размер S от 6 до 10 символов;
- при генерации исходной строки шанс $7/8$ сгенерировать подстроку из S случайных символов;
- при генерации исходной строки шанс $1/8$ сгенерировать искомую подстроку;
- на вход подается строка и искомая в ней подстрока;
- на выходе требуется получить файл с записями индексов вхождения подстроки.

2.2 Разработка алгоритмов

В листинге 1 рассмотрен псевдокод стандартного алгоритма поиска подстроки в строке.

Алгоритм 1 — Стандартный алгоритм

Входные данные: Строка *text*, подстрока *pattern*

Выходные данные: Индекс первого вхождения *pattern* в *text*, или -1 если не найден

```
1  n ← length(text);
2  m ← length(pattern);
3  цикл i ← 1 от n − m + 1 выполнять
4  |  j ← 1;
5  |  до тех пор, пока j ≤ m and text[i + j − 1] = pattern[j]
   |   выполнять
6  |  |  j ← j + 1;
7  |  конец
8  |  если j > m тогда
9  |  |  возвратить i;
10 |  конец
11 конец
12 возвратить -1;
```

На рисунке 1 приведена последовательная реализация программы.

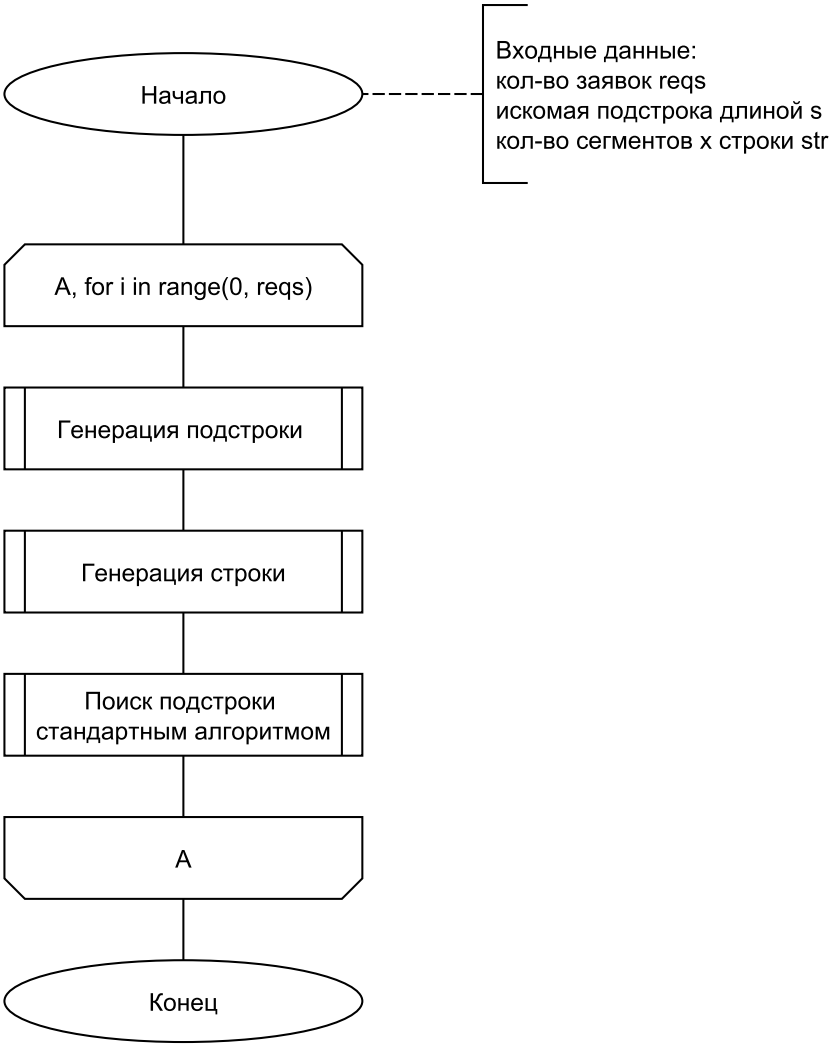

Рисунок 1 – Последовательная реализация

На рисунке 2 приведена реализация программы с двумя потоками.

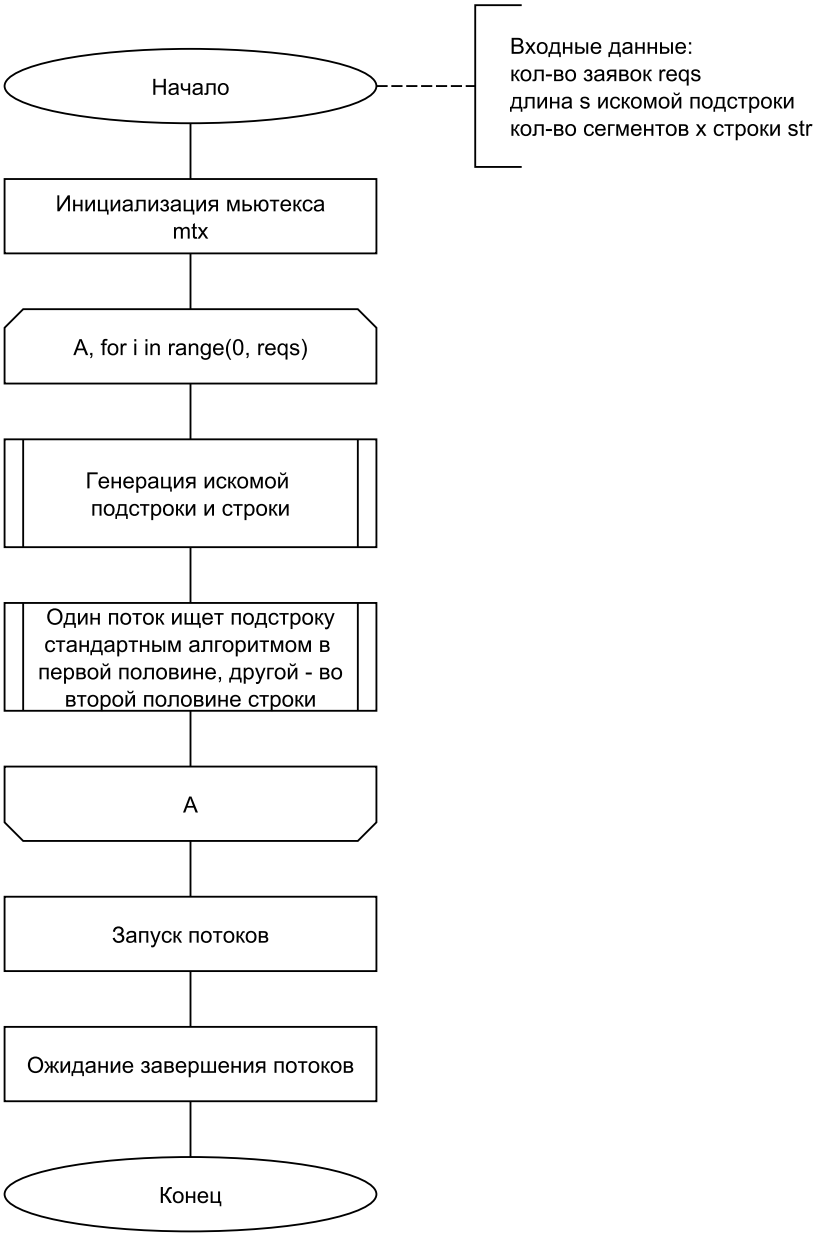

Рисунок 2 – Параллельная реализация

При этом, функция записи лога на рисунке 3 содержит критическую секцию, доступ к которой ограничивается мьютексом, который передается каждому потоку.

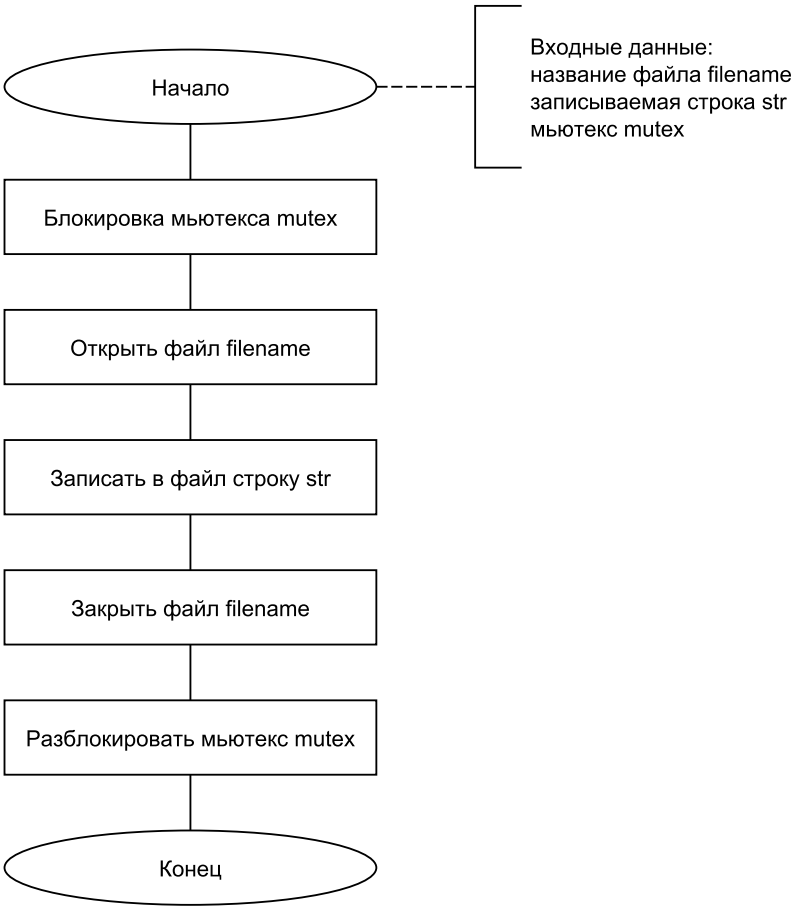

Рисунок 3 – Запись строки в файл

3 Технологическая часть

В данном разделе приведены требования к программному обеспечению, средства реализации и листинги кода.

3.1 Средства реализации

В качестве языка программирования для реализации лабораторной работы был выбран C++. Данный выбор обусловлен наличием функционала для работы с потоками thread и библиотекой chrono для замера времени выполнения алгоритмов.

3.2 Сведения о модулях программы

Программа состоит из следующих программных модулей:

- main.cpp — файл, содержащий меню и точку входа в программу;
- time.cpp — модуль замера времени выполнения алгоритмов;
- standart.cpp — модуль реализации стандартного алгоритма;
- generate.cpp — модуль генерации данных;
- implementation.cpp — модуль реализации двух видов алгоритма;
- соответствующие им файлы заголовков;
- plot.py — модуль создания графического представления значений.

3.3 Реализация алгоритмов

В листинге 1 приведена реализация стандартного алгоритма поиска подстроки в строке.

Листинг 1 – Стандартный алгоритм

```
void standart(string str, string sub, mutex &mtx, int shift) {
    size_t n = str.length();
    size_t s = sub.length();

    for (int i = 0; i < n - s; i++) {
        int j;
        for (j = 0; j < s; j++) {
            if (str[i + j] != sub[j])
                break;
        }
        if (j == s) {
            // cout << "Substring found at position " << i + shift
               << endl;
            write_in_file("log.txt", "Стандартный: " + sub + "
               найденавпозиции    " + to_string(i + shift), mtx);
        }
    }
}
```

В листингах 2 приведена последовательная реализация программы.

Листинг 2 – Последовательная реализация

```
void linear(int reqs, int s, int x) {
    mutex mtx;

  for (int i = 0; i < reqs; i++) {
    string sub = generate_substring(s);
    string str = generate_string(sub, x);

    standart(str, sub, mtx, 0);
  }
}
```

В листингах 3 приведена параллельная реализация программы.

Листинг 3 – Параллельная реализация

```
void parallel(int reqs, int s, int x) {
  vector<thread> threads;
    queue<pair<string, string>> q;
    mutex mtx;

    for (int i = 0; i < reqs; ++i) {
        string sub = generate_substring(s);
        string str = generate_string(sub, x);
        size_t n = str.length();
        string first_half = str.substr(0, n / 2 + s);
        string second_half = str.substr(n / 2, s);

        threads.emplace_back([&mtx, first_half, second_half, sub, n
           ]() {
            standart(first_half, sub, ref(mtx), 0);
            standart(second_half, sub, ref(mtx), n / 2);
            });
    }
    for (auto& thread : threads)
        thread.join();

    threads.clear();
}
```

В листинге 4 приведена реализация алгоритма генерации искомой подстроки.

Листинг 4 – Генерация подстроки

```
string generate_substring(int s) {
  string sub = "";

  for (int j = 0; j < s; j++)
    sub += 97 + rand() % 25;

  return sub;
}
```

В листинге 5 приведена реализация алгоритма генерации строки.

Листинг 5 – Генерация строки

```
string generate_string(string sub, int x) {
  srand(time(0));
  string str = "";
  int s = sub.length();

  for (int i = 0; i < x; i++) {
    int n = rand() % 8;

    if (n == 7) {
      str += sub[0] + sub[1];
      for (int j = 0; j < s; j++)
        str += sub[j];
    }
    else {
      for (int j = 0; j < s; j++)
        str += 97 + rand() % 25;
    }
  }

  return str;
}
```

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялся замерный эксперимент:

— операционная система Windows 11;

— память 16 ГБ;

— процессор 3,6 ГГц 6-ядерный процессор AMD Ryzen 5000 series 5.

Замеры проводились на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только интегрированной средой разработки и непосредственно выполняемой программой.

4.2 Пример работы программы

На рисунке 4 представлен пример работы программы.

```
Выберите необходимую задачу:
1 - запустить последовательную обработку
2 - запустить параллельную обработку
3 - замеры времени реализации
0 - выход

Ваш выбор: 1
Введите кол-во заявок, длину искомой подстроки, кол-во сегментов строки: 10 10 10000



Выберите необходимую задачу:
1 - запустить последовательную обработку
2 - запустить параллельную обработку
3 - замеры времени реализации
0 - выход

Ваш выбор: 2
Введите кол-во заявок, длину искомой подстроки, кол-во сегментов строки: 10 10 10000



Выберите необходимую задачу:
1 - запустить последовательную обработку
2 - запустить параллельную обработку
3 - замеры времени реализации
0 - выход

Ваш выбор: 0
```

Рисунок 4 – Пример работы программы

ПО не выводит сообщения в консоль, но пишет их в лог. На рисунке 5 приведен его пример.

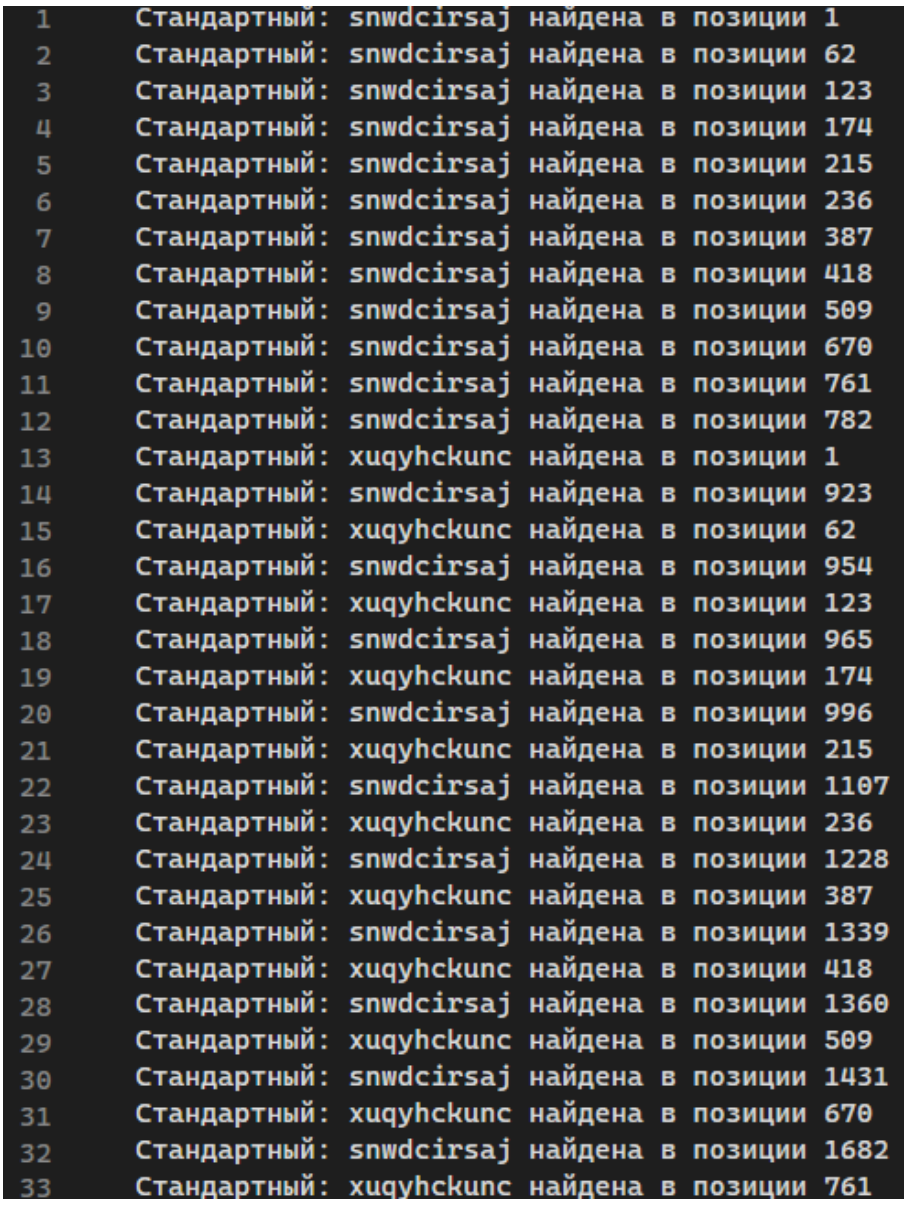

Рисунок 5 – Пример лога программы

4.3 Время выполнения реализованных алгоритмов

Замер времени проводился с помощью библиотеки *chrono* [6], в частности — с помощью структуры *steady_clock* [7].

Результаты измерений времени работы алгоритмов приведены на рисунке 6.

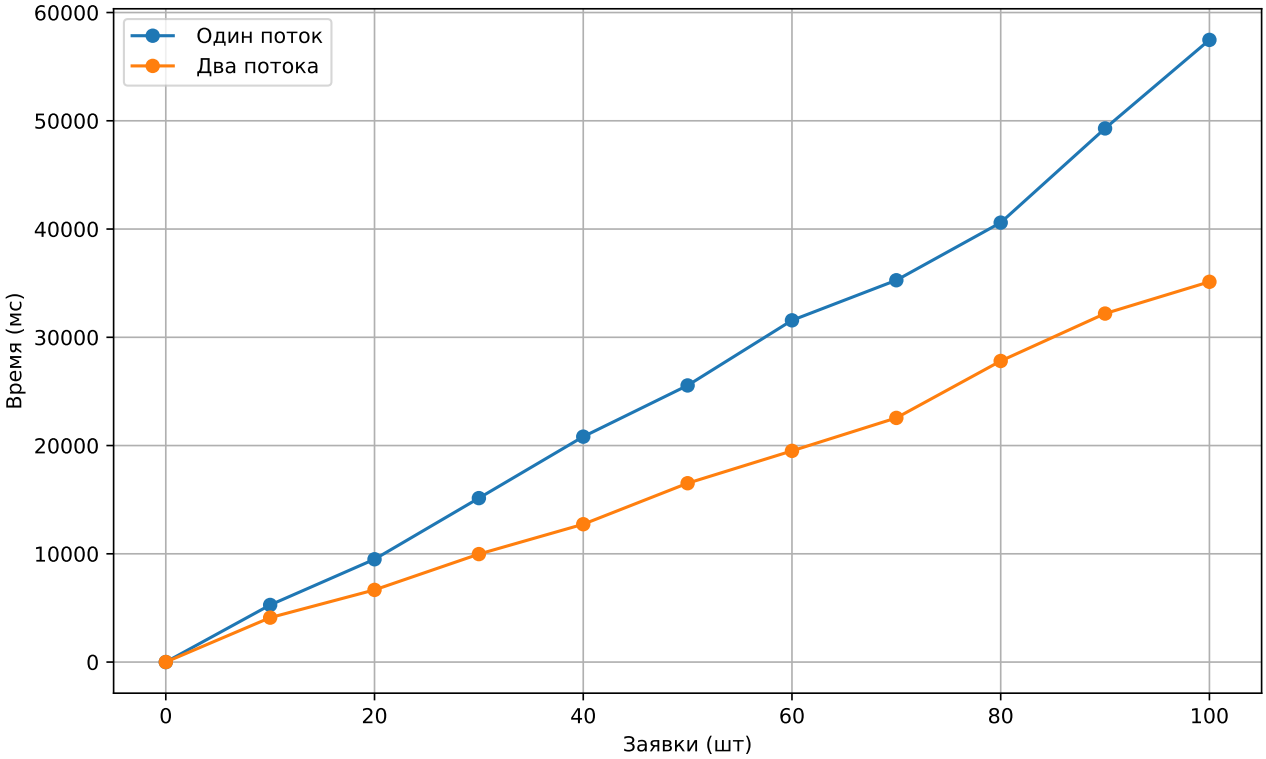

Рисунок 6 – Время выполнения алгоритмов

Время работы параллельной реализации алгоритма во всех случаях меньше его последовательной вариации.

Вывод

Во всех случаях последовательная реализация конвейера проигрывает по времени у параллельной. Исходя из линейности графиков, можно заключить, что при дальнейшем увеличении числа заявок параллельная реализация будет давать все больший «выигрышь» по времени.

ЗАКЛЮЧЕНИЕ

Цель работы достигнута: создано ПО с последовательной однопоточной и параллельной двупоточной реализацией нахождения подстроки в строке стандартным алгоритмом.

В ходе выполнения лабораторной работы были решены все задачи:

- описан стандартный алгоритм;
- определены средства программной реализации;
- реализован последовательный и параллельный вариант стандартного алгоритма;
- проведено сравнение обоих версий алгоритма;
- подготовлен отчет по лабораторной работе.

В результате исследования выяснилось, что для задачи поиска подстроки в строке с помощью стандартного алгоритма следует отдать предпочтение параллельной реализации, поскольку она быстрее последовательной при любом кол-ве заявок. В среднем выигрышь составляет 33.89% для кол-ва заявок до 100 штук.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Пелевин М.* Поиск подстроки в строке. — URL: `https://markoutte.me/students/substring`.
2. *Алексеенко А.* Информационная чувствительность алгоритма Кнута-Морриса-Пратта // Задачи системного анализа, управления и обработки информации. — 2010. — С. 5.
3. Архитектуры и топологии многопроцессорных вычислительных систем / А. Богданов [и др.] // М.: Интернет-университет информационных технологий. — 2004.
4. *Вирт Н.* Алгоритмы и структуры данных. Учебное пособие. — ДМК Пресс, 2010.
5. *Солдатова Г. П., Татаринов А. А., Болдырихин Н. В.* Основные алгоритмы поиска подстроки в строке // Academy. — 2018. — 5 (32). — С. 8—10.
6. *Microsoft.* <chrono>. — URL: `https://learn.microsoft.com/ru-ru/cpp/standard-library/chrono?view=msvc-170`.
7. *Microsoft.* Структура steady_clock. — URL: `https://learn.microsoft.com/ru-ru/cpp/standard-library/steady-clock-struct?view=msvc-170`.